Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions

Antoine Joux

DCSSI Crypto Lab
51, Bd de Latour-Maubourg
75700 Paris 07 SP, France
antoine.joux@m4x.org

Abstract. In this paper, we study the existence of multicollisions in iterated hash functions. We show that finding multicollisions, i.e. r-tuples of messages that all hash to the same value, is not much harder than finding ordinary collisions, i.e. pairs of messages, even for extremely large values of r. More precisely, the ratio of the complexities of the attacks is approximately equal to the logarithm of r. Then, using large multicollisions as a tool, we solve a long standing open problem and prove that concatenating the results of several iterated hash functions in order to build a larger one does not yield a secure construction. We also discuss the potential impact of our attack on several published schemes. Quite surprisingly, for subtle reasons, the schemes we study happen to be immune to our attack.

1 Introduction

One-Way hash functions are widely used cryptographic primitives, they operate on messages of almost arbitrary length[1] and output a fixed size value. Cryptographic hash functions should satisfy many security properties, such as the impossibility from a given hash to recover an associated message. However, the main security requirement for a hash function is its collision resistance. Informally, given a good hash function, no attacker should be able to find a pair of different messages M and M' leading to identical hash values. It is a well-known fact that all hash functions suffer from a generic birthday paradox based attack. More precisely, if H is a hash function that outputs n–bit values, then among the hash values of $2^{n/2}$ different messages, there exists a collision with non negligible probability. For this reason, hash functions that output values smaller than 160 bits are considered as deprecated. Yet, in the past, 128–bit hash functions were proposed and for legacy reasons they are still encountered in applications.

In practice, building a cryptographic function with an input of variable size is not a simple task. For this reason, most hash functions are based on an iterated construction that makes use of a so-called compression function, whose inputs have fixed sizes. Examples of such a construction are Snefru [7], MD4 [12], MD5 [13] or SHA [9]. In this paper, we specifically study one-way hash-functions built by iterating a compression function.

[1] The length is often bounded by a very large number such as 2^{64}. However, this is irrelevant for the attacks presented here.

Our main goal is to solve a long standing open problem: Is the concatenation of two independent hash values more secure than a single hash-value ? This question is of general interest and has appeared in many contexts. As far as we know, this construction first appeared as a generic transform in the PhD thesis of B. Preneel [10] and was called cascading. It was presented there as a mean to increase the security level at the cost of a decreased performance.

In fact, this idea of cascading hash functions is likely to be encountered in applications, for example, a construction called SHA-1x was used at some point in PGP and involves the computation of two SHA-1 values with a different set of initial constants. Similarly, the authors of RIPEMD [4] propose optional extensions of their hash functions to 256 and 320 bits values. In this case, the use of two hashing is extremely efficient since the original 128 and 160 bits algorithms already involve two parallel hashing whose results are normally added together.

Yet, according to [4], one should not expect to improve the security level with these constructions since unwanted dependencies between two slightly different instances of the same hash function may yield unforeseen attacks. In the same vein, a length doubling transform is suggested in the hash function chapter of [14], together with a warning that while no attacks are known several people have serious reservations about the construct.

As a consequence, the security of hash functions cascading is not very clear. Roughly, the cryptographic folklore states that the construction is good when two "independent" hash functions are cascaded. Clearly, this is true for random oracles and the generalization seems natural. For reference about this folklore knowledge, the interested reader may look up fact 9-27 in [6], that states that such a cascade is secure and that one could hope for a security of the order of the product of the security of the initial hash functions. However, we show in section 4 that this construction is in fact insecure, whenever an iterated hash function is involved in the cascading. Even cascading a 160-bit iterated hash function and a 160-bit random oracle does not really increase security above the initial 2^{80} level for collision resistance and above 2^{160} for preimage (or second preimage) resistance.

In order to solve this problem and prove that cascading two hash values is in fact insecure, we first address the simpler question of constructed multicollisions in an iterated hash function. This notion of multicollisions was first used by Merkle in [8] to study the security of a hash function based on DES. A related security property, namely r-collision freeness, has been suggested as a useful tool for building efficient cryptographic primitives. It was used for the micropayment scheme Micromint of Rivest and Shamir [11], for identification schemes by Girault and Stern in [5] and for signature schemes by Brickell and *al.* in [1]. The intuition behind this problem is that constructing r different messages with the same hash values should be much harder than constructing only two such messages. Once again, this is true when using random oracles. However, when iterated hash functions are involved, this intuition is false and multicollisions can be easily constructed.

The paper is organized as follows. In section 2 we recall some basic facts about iterated hash function and the possible security properties of hash functions. In section 3, we describe the basic attack for constructing multicollisions in iterated hash functions. In section 4, we use this attack as a tool and show that the security obtained when cascading several hash values is far from optimal. Unintuitively, this attack works even when two completely unrelated hash functions are cascaded and does not stem from any unforeseen correlation between similar hash functions. Finally, in section 5, we study the impact of our construction on several concrete schemes that rely on cascading or multicollision resistance. Very surprisingly, in all published examples, we encounter some obstruction which prevents the attack from working.

2 Basic Facts About Iterated Hash Functions

An iterated hash function H is built by iterating a basic compression function. The compression function f takes two inputs, a chaining variable and a message block, it outputs the next value of the chaining variable. Before processing, the message is first padded and split into elementary blocks. The padding itself is generally performed by appending a single '1' bit, followed by as many '0' bits as needed. To avoid some attacks, the binary encoding of the message length can also be added to complete the padding. This is called a Merkle-Damgard strengthening [8, 3]. Once the padded message is split into ℓ blocks, $M_1, \dots,$ M_ℓ, the chaining variable is set to some fixed initial value and the iteration is performed. To summarize, the hashing process works as follows:

– Pad the original message and split it into blocks $M_1, \dots, M_\ell$.
– Set H_0 to the initial value IV.
– For i from 1 to ℓ, let $H_i = f(H_{i-1}, M_i)$.
– Output $H(M) = H_\ell$.

Given such an iterated hash function, defining its security is a tricky matter. Ideally, the hash function is often seen as a concrete substitute for random oracles in cryptographic construction. Of course, it is well known (see [2]) that this extreme level of security is in fact impossible to reach. Thus, the security level of hash function is usually characterized by considering "easier" security goals. The most frequently encountered goal is the impossibility for a bounded adversary to find a collision in the hash function. We recall that a collision is a pair of different messages M and M' such that $H(M) = H(M')$. Due to the birthday paradox, there is a generic attack that find collisions after about $2^{n/2}$ evaluations of the hash function, where n is the size in bits of the hash values. The attack works by randomly choosing messages and computing their hash values until a collision occurs. Typically, with iterated hash functions, the size of messages' blocks is often larger than the size of the hash values themselves, and this attack usually works on the compression function itself. Other important security goals for hash functions are preimage resistance and second-preimage resistance. An attack against preimage resistance is an attack that, given some target value y, finds a message M such that $H(M) = y$. An attack against second preimage resistance, given a message M, finds another message such that $H(M) = H(M')$.

The best generic attacks against these security goals cost about 2^n evaluation of the function H.

The notion of collision can easily be generalized to that of r-way collision (or, for short, r-collision). A r-collision is simply a r-tuple of messages $M^{(1)}$, $\dots$, $M^{(r)}$, such that $H(M^{(1)}) = \cdots = H(M^{(r)})$. Assuming as above that the hash values behave almost randomly, finding an r-collision could be done by hashing about $2^{n\cdot(r-1)/r}$ messages. When r becomes large, this tends to 2^n. Due to this fact, relying on r-collision freeness in cryptographic construction seems a good way to gain more security without increasing the size of the hash functions. This is very tempting in some applications such as identification schemes [5] and signature schemes [1]. The next section demonstrates that, in fact, r-collisions in iterated hash functions are not much harder to construct than ordinary collisions, even for very large values of r.

3 Constructing Multicollisions

In this section, we show that constructing multicollisions in iterated hash function can be done quite efficiently. More precisely, constructing 2^t-collisions costs t times as much as building ordinary 2-collisions. Before describing the attack, let us remark that the padding process can be ignored as long as we consider collisions between messages of the same length. Indeed, in that case, the blocks of padding are identical. Moreover, if the intermediate hash chaining values collide at some point in the hash computation of two messages, the following values remain equal as soon as the ends of the messages are identical. Thus, on messages of the same length, collisions without the padding clearly lead to collisions with the padding.

For simplicity of exposure, we assume that the size of the message blocks is bigger than the size of the hash (and chaining) values. However, the attack can be easily generalized to the other case. We also assume that we can access a collision finding machine C, that given as input a chaining value h outputs two different blocks B and B' such that $f(h, B) = f(h, B')$. This collision finding machine may use the generic birthday attack or any specific attack based on a weakness of f. The most relevant property is that C should work properly for all chaining values[2]. To illustrate the basic idea, we first show how 4-collisions can be obtained with two calls to C. Starting from the initial value IV, we use a first call to C to obtain two different blocks, B_0 and B'_0 that yield a collision, i.e. $f(IV, B_0) = f(IV, B'_0)$. Let z denotes this common value and using a second call to C, find two other blocks B_1 and B'_1 such that $f(z, B_1) = f(z, B'_1)$. Putting these two steps together, we obtain the following 4-collision:

$$f(f(IV, B_0), B_1) = f(f(IV, B_0), B'_1) = f(f(IV, B'_0), B_1) = f(f(IV, B'_0), B'_1).$$

We now claim that this basic idea can be extended to much larger collisions by using more calls to the machine C. More precisely, using t calls, we can build 2^t-collisions in H. The attack works as follows:

[2] Or at least on a fixed proportion of them.

- Let h_0 be equal to the initial value IV of H.
- For i from 1 to t do:
  - Call C and find B_i and B'_i such that $f(h_{i-1}, B_i) = f(h_{i-1}, B'_i)$.
  - Let $h_i = f(h_{i-1}, B_i)$.
- Pad and output the 2^t messages of the form $(b_1, \ldots, b_t, \text{Padding})$ where b_i is one of the two blocks B_i or B'_i.

Clearly, the 2^t different messages built as above all reach the same final value. In fact, they have an even stronger property. Namely, all the intermediate hash values are equal, since all of the 2^t hashing processes go through h_0, h_1, ..., h_t. A schematic representation of these 2^t messages together with their common intermediate hash values is drawn in figure 1.

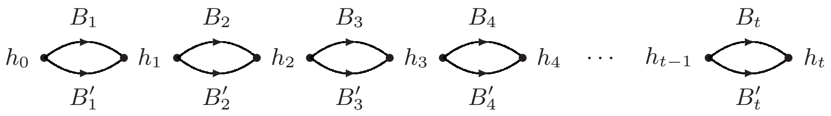

Fig. 1. Schematic representation of multicollision construction

Some generalizations. If f works on messages blocks which are smaller that the chaining values, the natural way to proceed is to group a few consecutive applications of f. For example, we can consider the function $f^{(2)}(h, B_1, B_2) = f(f(h, B_1), B_2)$ which composes two rounds of the compression function. As soon as the total size of the input blocks exceed the size of one chaining value, we can apply the original attack to the composed compression function.

Another generalization is to build 2^t-collisions from a 2-collision attack machine C that works only on a fixed proportion ϵ of the chaining values. Of course, this is not the case with the generic birthday attack, however, it may happen with some specific attacks. In that case, the basic attack described above only works with probability ϵ^t. Indeed, if any of the h_i does not belong to the set of chaining values that C can attack, we are in trouble. However, this bad behavior can be easily corrected by inserted a randomization step between two consecutive applications of C. Namely, after finding B_i and B'_i such that $f(h_{i-1}, B_i) = f(h_{i-1}, B'_i)$, choose a random block R_i and let:

$$h_i = f(f(h_{i-1}, B_i), R_i).$$

If h_i fails to be in the scope of C, change R_i to get another candidate. Altogether, this randomization technique leads to a global complexity of the attack of the order of t/ϵ calls to C.

4 On the Security of Cascaded Hash Functions

A natural construction to build large hash values is to concatenate several smaller hashes. For example, given two hash functions F and G, it seems reasonable given a message M to form the large hash value $(F(M)\|G(M))$. In this construction, F and G can either be two completely different hash functions or two slightly different instances of the same hash function[3]. If F and G are good iterated hash functions with no attack better than the generic birthday paradox attack, we claim that the hash function $F\|G$ obtained by concatenating F and G is not really more secure that F or G by itself. Moreover, this result applies both to collision resistance, preimage resistance and second preimage resistance.

4.1 Collision Resistance

Assume that F outputs an n_f-bit hash value and G an n_g-bit value. Then, with respect to collision resistance, the security level of F is $2^{n_f/2}$ and the level of G is $2^{n_g/2}$. If $F\|G$ was a good hash function, the complexity of the best attack would be $2^{(n_f+n_g)/2}$. We claim that there exists a much better attack which find collisions on $F\|G$ with complexity of the order of $n_g 2^{n_f/2} + 2^{n_g/2}$ if $n_f \leq n_g$ (respectively $n_f 2^{n_g/2} + 2^{n_f/2}$ if $n_f \geq n_g$). Assuming than $n_f \leq n_g$, the attack works as follows.

First, using the multicollision algorithm of section 3 with t equal to $n_g/2$ rounded up, construct a 2^t-collision on F. This costs t calls to the basic birthday paradox attack on the compression function of f, i.e. about $t2^{n_f/2}$ operations. This yields 2^t different messages with the same hash value on the F side. Since $t \geq n_g/2$, we can perform direct application of the birthday paradox on this set of 2^t elements and, with reasonable probability, expect that a collision occurs among the n_g-bit hashes of these 2^t messages by G. To increase the probability of success, it suffices to increase the value of t and add a few more calls to the basic attack on F.

Note that when evaluating the complexity of the attack, one must take into account the contribution of applying G to 2^t different messages of size t. With a naive implementation, this would cost $t2^t$ calls to the compression function of G. However, using the tree structure of the messages, this can be reduced to 2^t evaluations, assuming that the compression functions of F and G operate on the same size of blocks. Otherwise, it is necessary to add some padding between blocks in order to resynchronize the two functions.

A very important fact about this attack is that it does not require of G to be an iterative hash function. Any hash function will do, and this attack on cascaded hash works even when G is replaced by a random oracle[4]. Since a random oracle is independent from any function, this shows that the folklore knowledge about cascading hash functions is false. Thus, at least in that case, cascading two good and independent hash functions does not significatively improve collision resistance.

[3] E.g., two instances of SHA-1 with different constants.

[4] The only difference in that case is the fact that the evaluations of G on the 2^t messages can no longer be simplified. As a consequence, assuming that the cost of calling the random oracle G is linear in the size of the message, the contribution of G to the complexity becomes $t2^t$

4.2 Preimage and Second-Preimage Resistance

Concerning preimage resistance, it is already known that cascading two hash functions is, at least in some cases, the cascade is no stronger than the weakest hash function. Indeed, assume that we are hashing messages from a relatively small set, say a set of 2^m messages. Clearly, the best generic attack to find a preimage in that case is to perform exhaustive search on the set of messages, which costs 2^m steps. Assume that the output of each the two hash functions being cascaded is larger than m bit and that on this set of messages, one of the two hash functions, say F, has a shortcut attack. Then, we can clearly use this attack to recover a candidate preimage. Once this is done, it suffices to check that this candidate is also a preimage for the other function. The new attack presented in this section deals with a different case, where the entropy of the message space is much larger. It shows that even then the cascaded hash is no more secure than F itself.

Assume again that F outputs an n_f-bit hash value and G an n_g-bit value. Then, with respect to preimage resistance, the security level of F is 2^{n_f} and the level of G is 2^{n_g}. Indeed, the best known generic algorithm to break preimage resistance is to try random messages until the expected hash value is reached. This amounts to exhaustive search on the set of possible hash values. If $F\|G$ was a good hash function, the complexity of this exhaustive search attack would be $2^{(n_f+n_g)}$. As with collision resistance, there exists a much better attack which find a preimage on $F\|G$ with complexity of the order of $n_g 2^{n_f/2} + 2^{n_f} + 2^{n_g}$ if $n_f \leq n_g$ (respectively $n_f 2^{n_g/2} + 2^{n_g} + 2^{n_f}$ if $n_f \geq n_g$). Assuming than $n_f \leq n_g$, the attack works as follows.

First, using the multicollision algorithm of section 3 with t equal to n_g, construct a 2^t-collision on F. This costs t calls to the basic birthday paradox attack on the compression function of f, i.e. about $t 2^{n_f/2}$ operations. Then, search for an additional block that maps the last chaining value to the target value of F. Note that when looking for this additional block, we need to compute the output of the complete F function, including the padding of the message. However, this is a simple matter. After, this last step, we obtain 2^t different messages with the expected hash value on the F side. Since $t = n_g$, we expect that, with constant probability, at least one of these 2^t messages also match the expected t-bit value on the G side. Once again, the probability of success can be improved by adding a few more steps to the attack. Note that this attack on preimage resistance does not either require for G to be an iterative hash function. As before, it also works when G is replaced by a random oracle.

Clearly, the above attack finds a preimage for the target value which is essentially random. As a consequence, it can be applied directly without any change when a second preimage is requested.

4.3 Extensions and Open Problems

Given these attacks, it is natural to ask whether they generalizes to three or more concatenated hash values. In this section, we focus on the possibility of generalizing the collision search attack. We show that it does and that the generalization is almost straightforward. Indeed, assume that H is a third hash function on n_h bits, then using the above attack on $F\|G$ a couple of times, say $t \approx n_h/2$ times, it is possible as in section 3 to build a 2^t-collision on $F\|G$. Among these 2^t messages, we expect a collision of H. All in all, this yields a simultaneous collision on F, G and H. When $n_f = n_g = n_h = n$, the expression of the complexity simplifies and is of the order of $n^2 \cdot 2^{n/2}$. More generally, a simultaneous collision on q different n-bit iterative hash functions can be found with complexity $n^{q-1} \cdot 2^{n/2}$. Thus, the security of such a construction stays within a polynomial factor of the security of a single good iterative hash function of the same size. Similarly, variants of the attack on preimage resistance can be adapted to the case of q different hash functions. However, since they are more complicated, we do not present them here. One possible variant is described in appendix.

Another generalization of the above attack is also worth noting. In [14], B. Schneier described a different way of building a long hash from a hash function F. In this method, $F(M)$ is concatenated with $G(F(M)\|M)$ (or $G(M\|F(M))$). At first view, this is more complicated than the $F\|G$ construction. However, the very same attack can be applied. Indeed, when a 2^t-collision is found on $F(M)$, this fixes $F(M)$ in the first half of the big hash and also the copy of $F(M)$ in the call to G, thus a collision on the G part is expected exactly as before. The preimage attack also works as before. We leave open the problem of finding a related construction making q calls to n-bit hash function and with security higher than $n^{q-1} \cdot 2^{n/2}$, with respect to collision resistance.

One can also study a related question, how does the security of the concatenated hash $F\|G$ behaves, when F and G have non-generic attacks better than the birthday paradox collision search ? In that case, can $F\|G$ be significantly more secure than the best of F and G ?

For the sake of simplicity, assume once again that $n_f = n_g = n$. Then, if F has a collision finding algorithm C as in section 3 with complexity $2^{n/2}/n$ or better and G has no shortcut attack better than the birthday paradox, the security of $F\|G$ is essentially the same as the security of G itself. On the other hand, if G also admits a shortcut attack (as in section 3), it is unclear whether the two shortcut attacks may be used together to improve the composed attack against $F\|G$. Yet, some other type of attacks against G can be integrated into a better composed attack on $F\|G$. To give an example, let g denote the compression function of G. Assume that there exists a shortcut attack which given a large set $g_1, \ldots, g_N$ of chaining values finds a message block B and two indices i and j such that $g(g_i, B) = g(g_j, B)$ in time N. Clearly, such a merging attack could be used to turn an N-collision on F into a full collision on $F\|G$. Thus, it is safer to assume that $F\|G$ is essentially as secure as the best of F and G, no more.

5 Potential Applications

While the ideas of cascaded construction and multicollisions are frequently encountered in the cryptographic folklore, they are somewhat avoided in published papers. As a consequence, we were not able to find a single research paper that can be cryptanalyzed using the attacks presented here. In this section, we describe some published construction which were likely candidates and explain why the attacks failed.

Cascaded hash functions. Among the frequently encountered hash function, RIPEMD is the most suited to the cascaded construction. Indeed, the basic algorithm already consists of two separate hash computations which are put together at the end of the round function. Thus, using the result of the two chains to form a longer hash would be both natural and efficient. In fact, the authors of RIPEMD propose in [4] optional extensions to 256 and 320 bits by using this idea. These extensions are not fully specified, but a sketch is given. In this sketch, the authors of RIPEMD recommend to add to the basic cascade some interaction between the two parallel compression functions. More precisely, they propose to swap one register from the first chain and its counterpart in the second chain after each round of the compression function (5 rounds in RIPEMD-160 and 4 rounds in RIPEMD-128). This interaction was introduced as an additional security measure and, with respect to our attack, this countermeasure is very efficient and completely voids it.

Use of multicollisions. Among the published constructions that make use of multicollisions, one can cite the micropayment scheme Micromint [11], the identification scheme of Girault and Stern [5] and the signature scheme of Brickell and *al.* [1]. In these three applications, multicollisions are indeed used, however, in the proposed instances of the schemes, no iterated hash functions with a small internal memory is used. Instead, one encounters either a block cipher based compression function with a small block size but without iteration or a truncation of an iterated hash function with a relatively large output size. In both cases, our attack is unapplicable. In the first case, the required iteration is not available, in the second, the attack needs collisions on the full internal states of the hash function, rather than on the truncated states.

6 Conclusion

In this paper, we have shown that multicollisions in iterated hash functions are not really harder to find than ordinary collision. This yields the first effective attack against a natural construction that extend the size of hash values by concatenating several independent results. While considered suspect by some, especially when used with related hash functions, this construction had never been attacked before. The cryptanalysis we presented here yields attacks against collision resistance, preimage resistance and second preimage resistance. As a consequence, it leaves open the problem of constructing secure hash functions with variable-output length, which is a important primitive to instantiate some cryptographic paradigm such as the full domain hash.

Another important theoretical result is the fact that iterated hash functions cannot be used as entropy-smoothing functions on arbitrary sets of inputs. Devising good cryptographic entropy-smoothing functions would be a nice topic for future research.

References

1. E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. Design validation for siscrete logarithm based signature schemes. In *PKC'2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 276–292. Springer–Verlag, 2000.
2. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.
3. I. Damgård. A design principle for hash functions. In *Advances in Cryptology – Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer–Verlag, 1989.
4. H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160, a strengthened version of RIPEMD. In *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer–Verlag, 1996.
5. M. Girault and J. Stern. On the length of cryptographic hash-values used in identification schemes. In *Advances in Cryptology – Crypto'94*, volume 839 of *Lecture Notes in Computer Science*, pages 202–215. Springer–Verlag, 1994.
6. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997. Available on line : `http://www.cacr.math.uwaterloo.ca/hac`.
7. R. Merkle. A fast software one-way hash function. *Journal of Cryptology*, 3(1):43–58, 1990.
8. R. C. Merkle. One way hash functions and DES. In *Advances in Cryptology – Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer–Verlag, 1989.
9. Secure hash standard. Federal Information Processing Standard Publication 180–1, 1995.
10. B. Preneel. *Analysis and design of cryptographic hash functions.* PhD thesis, Katholieke Universiteit Leuven, January 1993.
11. R. Rivest and A. Shamir. PayWord and MicroMint – two simple micropayment schemes. *CryptoBytes*, 2(1):7–11, Spring 1996.
12. R. L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology – Crypto'90*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer–Verlag, 1991.
13. R. L. Rivest. The MD5 message-digest algorithm. Network Working Group Request for Comments: 1321, April 1992.
14. B. Schneier. *Applied Cryptography.* John Wiley & Sons, second edition edition, 1996.

A Preimage Resistance with Many Hash Functions Cascaded

While the attack against collision resistance described in section 4.1 is easily generalized to q hash functions and yields an attack with complexity $n^{q-1} \cdot 2^{n/2}$, assuming that each function outputs n bits, this is not the case for the attack on preimage resistance. Indeed, the attack we described in section 4.2 is not straightforward to generalize. The goal of this section is to present a variant of the attack that can be easily generalized. The drawback is that this variant is slightly more complicated than the initial attack. In this variant, each hash function is attacked in two steps. Each of these steps is constructed in a way that ensures compatibility with the previous hash functions. The first step within each hash function is to find two different sequences of blocks that, through the iterated hash process, sends the initial value back to itself. The cost of this amounts to twice exhaustive search on the set of possible chaining values. The second step is to find a terminating sequence of blocks that sends this chaining value to the target value for the current hash function. This costs about one exhaustive search on the same set.

When processing the first message, we look for single block sequences. Moreover, the terminating block should be correctly padded. A slight technical problem is that padding requires a priori knowledge of the final message size. However, we show at the end of this section that this size can be fixed in advance before launching the attack. Let T denotes this size, then we consider as inputs to the first hash functions the 2^{T-1} messages formed of $T-1$ blocks, each chosen among the two basic blocks that send the initial value h_0 to itself and one final block which sends h_0 to the target value h_F. A representation of these messages is given in figure reffig:preim

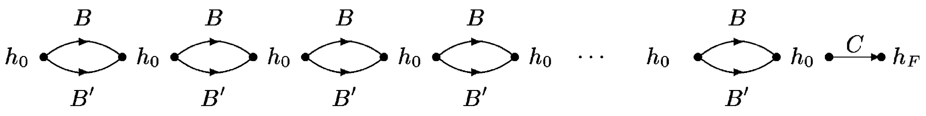

Fig. 2. First step of the preimage attack

With the second hash function, the looping sequences are constructed by concatenating n blocks chosen among the two (B and B') that makes the first hash function loop (a few more blocks can be added to increase the probability of finding good sequences). Clearly, when applying one of these sequences both the first and the second hash functions are going back to their initial values. The final sequence is constructed by concatenating many copies of the looping blocks B or B' and a single instance of the final block, in order to send the second hash function to its expected destination. Clearly, such a sequence also sends the first hash function to its target value. The advantage of this attack compared to that of section 4.2 is that additional hash functions can be processed by iterating the previous procedure. A notable exception is the computation of the last hash function which requires no looping part and can thus be simplified. The total runtime is clearly bounded by a polynomial ($O(n^q)$) times the cost of exhaustive search. The length T of the message can be easily predetermined and is of the order of n^{q-1}.